

MIIA0106
Python and C Programming Language

อาจารย์ สุทิศ องอาจ

ภาคการศึกษา 2 ปีการศึกษา 2567

MIIA0106	Python and C Programming Language		SUN-เช้า-23/03/2568
A	MON/2	D604	[CPA/1,CPAI/1,EMAT/1]#ALL#
	MON/3	MII203	
	MON/3	MII202 A	
	MON/3	MII202 B	
B	SAT/2	MII207 B	[CPA+/1,EMAT+/1,EMA/2]#ALL#
	SUN/1	MII202 A	
	SUN/1	MII202 B	
EP	- /0	-	

Update V1 2024-11-16

Update V2 2024-11-20

Update V3 2024-11-30

Update V4 2024-12-07

Update V5 2024-12-14

4. คาบที่ 5 Function

4.1. ความรู้พื้นฐานเกี่ยวกับฟังก์ชัน

4.1.1. ความหมายของฟังก์ชัน

ฟังก์ชันคือส่วนของโปรแกรมที่ใช้สำหรับดำเนินการเฉพาะเจาะจง โดยสามารถเรียกใช้งานได้ซ้ำในที่ต่าง ๆ ของโปรแกรม ช่วยลดการทำงานซ้ำซ้อนและเพิ่มความเข้าใจง่ายของโค้ด

4.1.2. ข้อดีของฟังก์ชัน:

- 1) ลดความซ้ำซ้อนในโค้ด (Code Reusability)
- 2) ช่วยแยกการทำงานออกเป็นส่วนย่อย (Modularity)
- 3) ทำให้โค้ดอ่านง่ายและแก้ไขได้สะดวก
- 4) สามารถทดสอบฟังก์ชันแต่ละส่วนได้แยกกัน (Unit Testing)

4.1.3. ข้อเสียของฟังก์ชัน:

- 1) อาจเพิ่มความซับซ้อนในการจัดการพารามิเตอร์
- 2) การเรียกใช้ฟังก์ชันซ้ำ ๆ อาจทำให้โปรแกรมใช้หน่วยความจำมากขึ้น

4.1.4. เหตุผลที่ต้องใช้ฟังก์ชัน:

- 1) เพื่อปรับปรุงโครงสร้างโปรแกรมให้เข้าใจง่าย
- 2) ลดเวลาในการพัฒนาโปรแกรมโดยใช้ฟังก์ชันซ้ำได้
- 3) ส่งเสริมการทำงานเป็นทีม เพราะสามารถแบ่งงานเป็นส่วนย่อยได้

4.2. ฟังก์ชันทั่วไป (Regular Function)

ฟังก์ชันที่สร้างขึ้นโดยผู้ใช้เพื่อทำงานเฉพาะ

C++

```
return_type function_name(parameters) {  
    // code block  
    return value; // (optional)  
}
```

รายละเอียด:

1. return_type

- ประเภทข้อมูลที่ฟังก์ชันจะคืนค่าให้ผู้เรียกใช้ (เช่น int, float, void, ฯลฯ)
- หากฟังก์ชันไม่คืนค่า ใช้ void

2. function_name

- ชื่อของฟังก์ชันที่ผู้เรียกใช้จะใช้เพื่ออ้างถึงฟังก์ชันนั้น
- ควรตั้งชื่อให้สื่อถึงการทำงานของฟังก์ชัน

3. parameters

- รายการพารามิเตอร์ (ถ้ามี) ที่ฟังก์ชันต้องการใช้เป็นข้อมูลนำเข้า
- สามารถมีได้หลายตัว แต่ละตัวระบุประเภทข้อมูลและชื่อ เช่น (int a, float b)
- ถ้าไม่มีพารามิเตอร์ ให้เว้นว่างไว้หรือใส่ ()

4. code block

- ชุดคำสั่งในฟังก์ชันที่ทำงานเมื่อถูกเรียกใช้งาน
- เขียนภายใน {}

5. return value (Optional)

- คำสั่ง return ใช้ส่งค่ากลับไปยังจุดที่เรียกใช้ฟังก์ชัน
- ถ้า return_type เป็น void ไม่ต้องใช้ return หรือใช้เพียง return; เพื่อออกจากฟังก์ชัน

ตัวอย่าง

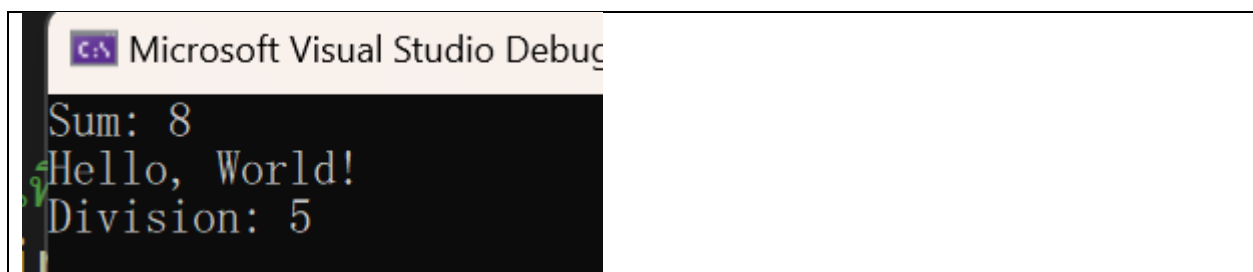
```
#include <iostream>
using namespace std;

// ฟังก์ชันที่คืนค่าจำนวนเต็ม
int add(int x, int y) {
    return x + y; // ส่งผลลัพธ์ของ x + y กลับไป
}

// ฟังก์ชันที่ไม่คืนค่า
void printHello() {
    cout << "Hello, World!" << endl;
}

// ฟังก์ชันที่คืนค่าจำนวนจริง
double divide(double a, double b) {
    if (b != 0)
        return a / b; // ส่งค่าผลหาร
    else
        return 0.0; // กรณีหารด้วยศูนย์
}

int main() {
    cout << "Sum: " << add(3, 5) << endl;
    printHello();
    cout << "Division: " << divide(10.0, 2.0) << endl;
    return 0;
}
```

The image shows a screenshot of the Microsoft Visual Studio Debug Console. The window title is "Microsoft Visual Studio Debug Console". The output text is displayed on a black background with white and green text. The output consists of three lines: "Sum: 8", "Hello, World!", and "Division: 5". The text "Hello, World!" is highlighted with a green selection bar on the left side of the console window.

```
Sum: 8
Hello, World!
Division: 5
```

python

```
def function_name(parameters):  
    # code block  
    return value
```

v

คำอธิบาย:

1. ฟังก์ชัน add:
 - รับค่าพารามิเตอร์สองตัว x และ y
 - คืนค่าผลรวมของทั้งสอง
2. ฟังก์ชัน print_hello:
 - ฟังก์ชันนี้ไม่มีการคืนค่า (void ใน C++) และใช้แสดงข้อความ "Hello, World!"
3. ฟังก์ชัน divide:
 - รับค่าพารามิเตอร์สองตัว a และ b
 - หาก b ไม่เท่ากับ 0 จะคืนค่าผลหาร a / b
 - หาก b เท่ากับ 0 จะคืนค่า 0.0 เพื่อหลีกเลี่ยงการหารด้วยศูนย์
4. if `__name__ == "__main__":`:
 - ส่วนนี้ใช้สำหรับทำให้โค้ดในไฟล์นี้สามารถรันเป็นโปรแกรมหลัก หรือเป็นโมดูลที่นำไปใช้ในไฟล์อื่นได้

4.3. ฟังก์ชันแบบ Inline (Inline Function)

ฟังก์ชันขนาดเล็กที่คอมไพเลอร์พยายาม "ขยาย" เข้าไปแทนที่การเรียกฟังก์ชันเพื่อเพิ่มประสิทธิภาพ

4.3.1. คุณสมบัติเด่นของ Inline Function:

- 1) ลด Overhead ของการเรียกฟังก์ชัน (Function Call) โดยแทรกโค้ดของฟังก์ชันในจุดที่เรียกใช้
- 2) เหมาะสำหรับฟังก์ชันขนาดเล็กหรือการคำนวณซ้ำ ๆ
- 3) ไม่เหมาะกับฟังก์ชันขนาดใหญ่ เนื่องจากทำให้ขนาดไฟล์ที่คอมไพล์ (Binary File) ใหญ่ขึ้น

โครงสร้าง:

```
inline return_type function_name(parameters) {  
    // code block  
    return value; // (optional)  
}
```

รายละเอียดของแต่ละส่วน:

1. inline

- คำสั่ง inline บอกคอมไพเลอร์ให้ขยายโค้ดของฟังก์ชันโดยตรงในที่ที่ฟังก์ชันถูกเรียกใช้งาน แทนที่จะเรียกไปยังตำแหน่งหน่วยความจำของฟังก์ชันนั้น
- เหมาะสำหรับฟังก์ชันขนาดเล็กที่เรียกใช้งานบ่อย เพื่อประหยัดค่าใช้จ่ายในการเรียกฟังก์ชัน (ลด Overhead)

2. return_type

- ประเภทของค่าที่ฟังก์ชันจะคืนค่าให้กับผู้เรียกใช้ เช่น int, float, void, เป็นต้น
- ถ้าไม่มีค่าคืนกลับ ให้ใช้ void

3. function_name

- ชื่อของฟังก์ชันที่อธิบายการทำงาน
- ใช้สำหรับเรียกฟังก์ชันจากส่วนอื่นของโปรแกรม

4. parameters

- พารามิเตอร์คือค่าที่ส่งเข้ามาในฟังก์ชันเพื่อใช้ในการประมวลผล

- สามารถมีได้หลายตัว เช่น (int x, double y) หรือไม่มีพารามิเตอร์เลยก็ได้ ()

5. code block

- ชุดคำสั่งที่ต้องการให้ฟังก์ชันทำงาน
- อยู่ภายใน {}

6. return value (ถ้ามี)

- ใช้คำสั่ง return เพื่อคืนค่ากลับไปยังจุดที่เรียกใช้งาน
- ถ้า return_type เป็น void ไม่จำเป็นต้องมี return

ตัวอย่าง

```
#include <iostream>
using namespace std;

inline int square(int x) {
    return x * x;
}

inline void printMessage() {
    cout << "This is an inline function." << endl;
}

inline double multiply(double a, double b) {
    return a * b;
}

int main() {
    cout << "Square of 4: " << square(4) << endl;
    printMessage();
    cout << "Product: " << multiply(2.5, 4.1) << endl;
    return 0;
}
```


4.4. ฟังก์ชันแบบ Overloaded (Function Overloading)

ฟังก์ชันที่มีชื่อเดียวกัน แต่พารามิเตอร์แตกต่างกันโครงสร้าง

สร้าง:

```
return_type function_name(parameters);  
return_type function_name(parameters, ...);
```

โครงสร้าง:

```
return_type function_name(parameters);  
ฟังก์ชันที่รับพารามิเตอร์หนึ่งรูปแบบ  
return_type function_name(parameters, ...);  
ฟังก์ชันที่รับพารามิเตอร์อีกแบบที่แตกต่างกันในจำนวนหรือชนิด
```

รายละเอียด:

1. **return_type**
 - ประเภทข้อมูลที่ฟังก์ชันคืนค่า เช่น int, double, string เป็นต้น
 - ฟังก์ชัน Overloading สามารถมีประเภทการคืนค่าต่างกันได้ แต่พารามิเตอร์ต้องไม่ซ้ำกัน
2. **function_name**
 - ชื่อของฟังก์ชันที่เหมือนกันในทุกกรณี
3. **parameters**
 - รายการพารามิเตอร์สามารถ:
 - ต่างกันในจำนวน เช่น (int x) กับ (int x, int y)
 - ต่างกันในชนิด เช่น (int x) กับ (double x)
 - ต่างกันในลำดับ เช่น (int x, double y) กับ (double x, int y)
4. **การทำงานของคอมไพเลอร์:**
 - คอมไพเลอร์จะแยกฟังก์ชันที่มีชื่อเดียวกันโดยพิจารณาจาก "ลายเซ็นของฟังก์ชัน" ซึ่งประกอบด้วย:
 - ชื่อฟังก์ชัน
 - จำนวนและชนิดของพารามิเตอร์

ตัวอย่าง

```
#include <iostream>
using namespace std;

// ฟังก์ชันที่รับจำนวนเต็ม
void printValue(int x) {
    cout << "Integer value: " << x << endl;
}

// ฟังก์ชันที่รับจำนวนจริง
void printValue(double x) {
    cout << "Double value: " << x << endl;
}

// ฟังก์ชันที่รับข้อความ
void printValue(string str) {
    cout << "String value: " << str << endl;
}

int main() {
    printValue(10);           // เรียกใช้ฟังก์ชันที่รับ int
    printValue(3.14);         // เรียกใช้ฟังก์ชันที่รับ double
    printValue("Hello");      // เรียกใช้ฟังก์ชันที่รับ string
    return 0;
}
```

กรณีเพิ่มเติม (พารามิเตอร์หลายตัว):

```
#include <iostream>
using namespace std;

// ฟังก์ชันที่รับ 1 พารามิเตอร์
int sum(int x) {
    return x;
}

// ฟังก์ชันที่รับ 2 พารามิเตอร์
int sum(int x, int y) {
    return x + y;
}

// ฟังก์ชันที่รับ 3 พารามิเตอร์
int sum(int x, int y, int z) {
    return x + y + z;
}

int main() {
    cout << "Sum (1 parameter): " << sum(5) << endl;
    cout << "Sum (2 parameters): " << sum(3, 7) << endl;
    cout << "Sum (3 parameters): " << sum(1, 2, 3) << endl;
    return 0;
}
```

4.5. ฟังก์ชันแบบ Template (Template Function)

ฟังก์ชันที่กำหนดชนิดข้อมูล (Data Type) ไว้แบบทั่วไป เพื่อรองรับการทำงานกับหลายชนิดข้อมูลโดยไม่ต้องเขียนโค้ดซ้ำ

โครงสร้าง:

```
template <typename T>
T function_name(T parameter) {
    // code block
    return value;
}
```

```
#include <iostream>
using namespace std;

template <typename T>
T add(T a, T b) {
    return a + b;
}

int main() {
    cout << "Integer sum: " << add(5, 10) << endl;    // ใช้งานกับ int
    cout << "Double sum: " << add(3.5, 2.7) << endl;  // ใช้งานกับ double
    return 0;
}
```

4.6. ฟังก์ชันแบบ Recursive (Recursive Function)

ฟังก์ชันที่เรียกตัวเองซ้ำ ๆ เพื่อแก้ปัญหาแบบวนซ้ำจนได้ผลลัพธ์สุดท้าย

โครงสร้าง:

```
return_type function_name(parameters) {  
    if (base_condition) {  
        return base_value; // กรณีสิ้นสุด  
    }  
    return function_name(updated_parameters); // เรียกตัวเอง  
}
```

```
#include <iostream>  
using namespace std;  
  
int factorial(int n) {  
    if (n == 0) return 1; // เงื่อนไขหยุด  
    return n * factorial(n - 1); // เรียกตัวเอง  
}  
  
//5!  
int main() {  
    cout << "Factorial of 5: " << factorial(5) << endl;  
    return 0;  
}
```

```
def function_name(parameters):  
    if base_condition:  
        return base_value  
    return function_name(updated_parameters)
```

```
def factorial(n):  
    if n == 0:  
        return 1  
    return n * factorial(n - 1)  
  
print("Factorial of 5:", factorial(5))
```

การเขียนโปรแกรมคำนวณ Factorial แบบปกติ (Non-Recursive Function) คือการใช้ ลูป เพื่อคำนวณค่า Factorial โดยไม่เรียกฟังก์ชันตัวเองซ้ำ ซึ่งสามารถอธิบายให้นักศึกษาเข้าใจง่ายขึ้นดังนี้:

โค้ดตัวอย่างแบบ Non-Recursive Function

```
int factorial_NonRecursive(int n) {
    int result = 1; // ค่าเริ่มต้น
    for (int i = 1; i <= n; i++) {
        result *= i; // คูณค่าของ result ด้วย i ทีละขั้น
        cout << "Factorial of : " << i << "=" << result << endl;
    }
    return result;
}
```

เปรียบเทียบการทำงานระหว่าง Recursive และ Non-Recursive		
หัวข้อ	Recursive Function	Non-Recursive Function
โครงสร้าง	เรียกตัวเองซ้ำจนถึงเงื่อนไขหยุด	ใช้ลูปในการคำนวณ
เหมาะสมกับ	ปัญหาที่มีโครงสร้างซ้อนกัน หรือซับซ้อน	การคำนวณทั่วไปหรือโครงสร้างง่ายๆ
ความซับซ้อนของโค้ด	อาจดูยากสำหรับมือใหม่	เข้าใจง่ายและอ่านง่ายกว่า
หน่วยความจำ (Memory)	ใช้ Stack ในการเก็บสถานะ	ใช้หน่วยความจำคงที่ (ไม่ต้องใช้ Stack)

4.7. ฟังก์ชันที่เป็นสมาชิกของคลาส (Member Function)

ฟังก์ชันที่อยู่ภายในคลาสและใช้สำหรับจัดการข้อมูลหรือพฤติกรรมของคลาสนั้น ๆ

```
class ClassName {  
public:  
    return_type function_name(parameters) {  
        // code block  
    }  
};
```

```
#include <iostream>  
using namespace std;  
  
class Calculator {  
public:  
    int add(int a, int b) {  
        return a + b;  
    }  
    int multiply(int a, int b) {  
        return a * b;  
    }  
};  
  
int main() {  
    Calculator calc;  
    cout << "Sum: " << calc.add(3, 4) << endl;  
    cout << "Product: " << calc.multiply(5, 6) << endl;  
    return 0;  
}
```

4.8. ฟังก์ชันแบบ Lambda (Lambda Function)

ฟังก์ชันแบบนิพจน์ที่ไม่มีชื่อ ใช้สำหรับงานเฉพาะที่ไม่ต้องการสร้างฟังก์ชันแบบปกติ

โครงสร้าง:

```
[ capture_list ] ( parameters ) -> return_type {  
    // code block  
};
```

```
#include <iostream>  
using namespace std;  
  
int main() {  
    auto add = [](int a, int b) -> int {  
        return a + b;  
    };  
    cout << "Sum using Lambda: " << add(3, 5) << endl;  
    return 0;  
}
```

4.9. ฟังก์ชันที่เป็นเพื่อนของคลาส (Friend Function)

ฟังก์ชันที่สามารถเข้าถึงสมาชิก private/protected ของคลาสได้โดยไม่ต้องเป็นสมาชิกของคลาสนั้น

โครงสร้าง

```
class ClassName {  
    friend return_type function_name(parameters);  
};
```

```
#include <iostream>  
using namespace std;  
  
class Box {  
private:  
    int width;  
public:  
    Box(int w) : width(w) {}  
    friend int getWidth(Box b); // ประกาศฟังก์ชันเป็นเพื่อน  
};  
  
int getWidth(Box b) {  
    return b.width; // เข้าถึงสมาชิก private ได้  
}  
  
int main() {  
    Box box(10);  
    cout << "Box width: " << getWidth(box) << endl;  
    return 0;  
}
```


4.10. ฟังก์ชันแบบ Virtual (Virtual Function)

ฟังก์ชันในคลาสฐานที่สามารถถูก Override โดยคลาสที่สืบทอด เพื่อเปลี่ยนพฤติกรรมในคลาสลูก

โครงสร้าง:

```
class Base {  
public:  
    virtual return_type function_name(parameters) {  
        // base code block  
    }  
};
```

```
#include <iostream>  
using namespace std;  
  
class Base {  
public:  
    virtual void display() {  
        cout << "Display from Base class" << endl;  
    }  
};  
  
class Derived : public Base {  
public:  
    void display() override {  
        cout << "Display from Derived class" << endl;  
    }  
};  
  
int main() {  
    Base* basePtr;  
    Derived obj;  
    basePtr = &obj;  
    basePtr->display(); // แสดงผลจาก Derived class  
    return 0;  
}
```

4.11. สรุปประเภทของฟังก์ชันใน C++ ทั้ง 9 แบบ

ประเภทฟังก์ชัน	คำอธิบาย	ตัวอย่างการใช้งาน
1. Regular Function	ฟังก์ชันทั่วไปที่ผู้ใช้สร้างขึ้นเพื่อทำงานเฉพาะ	<code>int add(int a, int b) { return a + b; }</code>
2. Inline Function	ฟังก์ชันขนาดเล็กที่คอมไพเลอร์แทรกโค้ดตรงที่เรียกใช้ เพื่อลด Overhead	<code>inline int square(int x) { return x * x; }</code>
3. Function Overloading	ฟังก์ชันที่มีชื่อเดียวกัน แต่พารามิเตอร์ต่างกัน	<code>void print(int x); void print(double x);</code>
4. Template Function	ฟังก์ชันที่กำหนดชนิดข้อมูลแบบทั่วไป เพื่อรองรับหลายประเภทข้อมูล	<code>template <typename T> T add(T a, T b);</code>
5. Recursive Function	ฟังก์ชันที่เรียกตัวเองซ้ำ ๆ จนกว่าจะถึงเงื่อนไขสิ้นสุด	<code>int factorial(int n) { return n == 0 ? 1 : n * factorial(n-1); }</code>
6. Member Function	ฟังก์ชันที่เป็นสมาชิกของคลาสและจัดการข้อมูลของคลาส	<code>class MyClass { void display(); };</code>
7. Lambda Function	ฟังก์ชันนิพจน์ที่ไม่มีชื่อ ใช้สำหรับงานเฉพาะจุด	<code>auto add = [](int a, int b) { return a + b; };</code>
8. Friend Function	ฟังก์ชันที่สามารถเข้าถึงสมาชิก private/protected ของคลาสโดยไม่ต้องเป็นสมาชิกของคลาส	<code>friend int getWidth(Box b);</code>
9. Virtual Function	ฟังก์ชันในคลาสนั้นที่สามารถ Override พฤติกรรมในคลาสลูก	<code>virtual void display();</code>

คำอธิบายเพิ่มเติม:

- 1) Regular Function: ฟังก์ชันทั่วไปที่ทำงานตามที่คุณเขียนกำหนด เช่น การคำนวณหรือแสดงผล
- 2) Inline Function: ช่วยเพิ่มประสิทธิภาพสำหรับฟังก์ชันขนาดเล็กโดยการลดการเรียกฟังก์ชัน
- 3) Function Overloading: ใช้ชื่อฟังก์ชันเดียวกัน แต่สามารถทำงานได้หลากหลายรูปแบบโดยขึ้นอยู่กับพารามิเตอร์
- 4) Template Function: ช่วยลดการเขียนโค้ดซ้ำเมื่อทำงานกับข้อมูลหลายประเภท
- 5) Recursive Function: ใช้แก้ปัญหาที่มีรูปแบบซ้ำ ๆ เช่น การคำนวณ Factorial หรือ Fibonacci
- 6) Member Function: ฟังก์ชันที่เป็นส่วนหนึ่งของคลาสในโปรแกรมแบบเชิงวัตถุ (OOP)
- 7) Lambda Function: ฟังก์ชันที่ไม่มีชื่อ เหมาะสำหรับงานที่ต้องการโค้ดสั้นกระชับ
- 8) Friend Function: ฟังก์ชันที่สามารถเข้าถึงข้อมูลภายในคลาสโดยไม่ต้องเป็นสมาชิกของคลาส
- 9) Virtual Function: ใช้ใน OOP เพื่อสนับสนุน Polymorphism (การทำงานที่แตกต่างกันในคลาสลูก)

4.12. ปฏิบัติ (3 ชม)

วิเคราะห์โค้ดและเรียนรู้การทำงานของ Recursive Function ผ่านโปรแกรม

โครงสร้างของโปรแกรม โปรแกรมนี้มีการคำนวณ Factorial ด้วยหลากหลายวิธี:

- 1) factorial: ใช้ Recursive Function แบบสมบูรณ์ (พื้นฐาน)
- 2) factorial_NonRecursive: ใช้ ลูป (Non-Recursive) ในการคำนวณ
- 3) factorial0, factorial1, และ factorial2: เป็นขั้นตอนการพัฒนา Recursive Function

```
#include <iostream>
using namespace std;

int factorial(int n) {
    cout << "Factorial of : " << n << endl;

    if (n == 0) return 1; // เงื่อนไขหยุด

    int result = n * factorial(n - 1); // คำนวณ Factorial
    cout << n << " * factorial(" << n - 1 << ") = " << result << endl; // แสดงผลคูณ
    return result;
}

int factorial_NonRecursive(int n) {
    int result = 1; // ค่าเริ่มต้น
    for (int i = 1; i <= n; i++) {

        result *= i; // คูณค่าของ result ด้วย i ทีละขั้น
        cout << "Factorial of : " << i << "=" << result << endl;
    }
    return result;
}

int factorial0(int n) {
    cout << "Factorial of : " << n << endl;

    if (n == 0) return 1;

    int result = n;
    cout << n << " * factorial(" << n - 1 << ") = " << result << endl;
    return result;
}

int factorial1(int n) {
    cout << "Factorial of : " << n << endl;

    if (n == 0) return 1;
```

```

        int result = n * factorial0(n - 1);
        cout << n << " * factorial(" << n - 1 << ") = " << result << endl;
        return result;
    }

    int factorial2(int n) {
        cout << "Factorial of : " << n << endl;

        if (n == 0) return 1;
        int result = n;
        result = n * factorial1(n - 1);
        cout << n << " * factorial(" << n - 1 << ") = " << result << endl;
        return result;
    }

    int main() {
        cout << "Factorial of 5: " << factorial(5) << endl;
        cout << "factorial_NonRecursive of 5: " << factorial_NonRecursive(5) <<
endl;

        cout << "Factorial of 2: " << factorial2(2) << endl;
        return 0;
    }
}

```

สรุปความเข้าใจของโปรแกรม

สรุป

- ฟังก์ชัน **Recursive** (factorial) ใช้การเรียกตัวเองจนถึงเงื่อนไขหยุด
- ฟังก์ชัน **Non-Recursive** (factorial_NonRecursive) ใช้ลูป for เพื่อลดการใช้หน่วยความจำ
- การซ้อนฟังก์ชัน (factorial2, factorial1, factorial0) เป็นการแสดงตัวอย่างการจัดการลำดับการเรียกฟังก์ชัน

ทั้ง Recursive และ Non-Recursive มีผลลัพธ์สุดท้ายเหมือนกัน แต่ Recursive อาจซับซ้อนในแง่ของการจัดการหน่วยความจำเมื่อ n มีค่าสูงมาก!

อธิบายแบบละเอียด

ฟังก์ชันในโค้ด

1. factorial(int n)

- เป็นฟังก์ชัน **Recursive** สำหรับคำนวณ Factorial
- หาก $n == 0$ จะคืนค่า 1 เป็นเงื่อนไขหยุด

- หาก $n > 0$ จะคูณ n กับ $\text{factorial}(n - 1)$ แล้วส่งค่าผลลัพธ์กลับ
- แสดงขั้นตอนการคำนวณในแต่ละระดับ

การทำงาน (ตัวอย่างสำหรับ $\text{factorial}(5)$):

1. $5 \times \text{factorial}(4)$
2. $4 \times \text{factorial}(3)$
3. $3 \times \text{factorial}(2)$
4. $2 \times \text{factorial}(1)$
5. $1 \times \text{factorial}(0) \rightarrow 1$ (เงื่อนไขหยุด)

2. $\text{factorial_NonRecursive}(\text{int } n)$

- เป็นฟังก์ชันแบบ **Non-Recursive** สำหรับคำนวณ Factorial โดยใช้ลูป for
- เริ่มจาก $\text{result} = 1$ แล้ววนลูปคูณค่าทีละขั้นจาก 1 ถึง n
- แสดงค่าผลลัพธ์ที่เปลี่ยนแปลงในแต่ละรอบของลูป

การทำงาน (ตัวอย่างสำหรับ $\text{factorial_NonRecursive}(5)$):

1. $\text{result} = 1$
2. $\text{result} = 1 \times 1 = 1$
3. $\text{result} = 1 \times 2 = 2$
4. $\text{result} = 2 \times 3 = 6$
5. $\text{result} = 6 \times 4 = 24$
6. $\text{result} = 24 \times 5 = 120$

3. $\text{factorial0}(\text{int } n)$, $\text{factorial1}(\text{int } n)$, $\text{factorial2}(\text{int } n)$

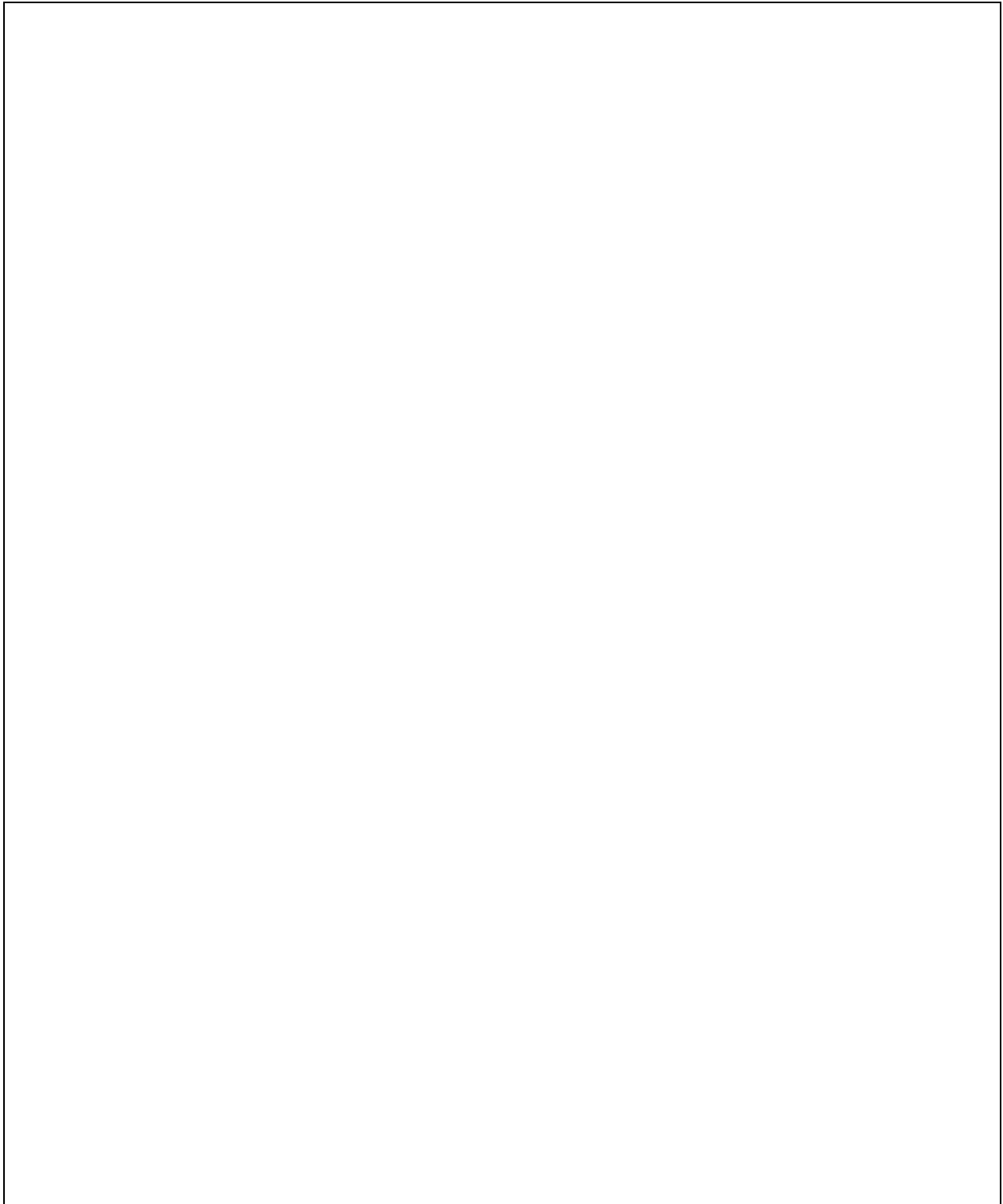
- เป็นฟังก์ชันที่ซ้อนกันและเชื่อมโยงการคำนวณแบบ **Recursive** ระหว่างฟังก์ชัน
- factorial2 จะเรียก factorial1 และ factorial1 จะเรียก factorial0
- factorial0 ทำงานคล้าย factorial ดั้งเดิม แต่ไม่เรียกตัวเองอีก

การทำงาน (ตัวอย่างสำหรับ $\text{factorial2}(2)$):

1. $\text{factorial2}(2)$ เรียก $\text{factorial1}(1)$
2. $\text{factorial1}(1)$ เรียก $\text{factorial0}(0)$

3. `factorial0(0)` คืนค่า 1 (เงื่อนไขหยุด)
4. `factorial1(1)` คูณ 1 กับผลลัพธ์จาก `factorial0` (ได้ 1)
5. `factorial2(2)` คูณ 2 กับผลลัพธ์จาก `factorial1` (ได้ 2)

ผลการรันโปรแกรม



เขียนโปรแกรมที่มีฟังก์ชันดังต่อไปนี้:

1. ล็อกอิน:

เมื่อเปิดโปรแกรมให้ผู้ใช้งานกรอก **ชื่อผู้ใช้** และ **รหัสผ่าน** โดยค่าที่ถูกต้องคือ "admin" สำหรับทั้งชื่อผู้ใช้และรหัสผ่าน หากกรอกผิดให้แจ้งว่า "ชื่อผู้ใช้หรือรหัสผ่านไม่ถูกต้อง" และจบโปรแกรม

2. เมนูหลัก:

หลังล็อกอินสำเร็จ จะแสดงเมนูให้ผู้เลือกใช้ทำงานดังนี้:

- 1: **บวกเลข** (รับเลขสองจำนวนจากผู้ใช้และแสดงผลรวม)
- 2: **ลบเลข** (รับเลขสองจำนวนจากผู้ใช้และแสดงผลต่าง)
- 3: **คูณเลข** (รับเลขสองจำนวนจากผู้ใช้และแสดงผลคูณ)
- 4: **หารเลข** (รับเลขสองจำนวนจากผู้ใช้และแสดงผลหาร หากหารด้วย 0 ให้แสดงข้อความเตือน)
- 5: **หาพื้นที่สี่เหลี่ยม** (รับค่าความกว้างและความยาวจากผู้ใช้ แล้วแสดงพื้นที่)
- 6: **วนลูปแสดงเลข 1 ถึง N โดยใช้ For Loop**
(รับค่าจำนวนเต็ม N จากผู้ใช้ และแสดงตัวเลขจาก 1 ถึง N)
- 7: **วนลูปแสดงเลข 1 ถึง N โดยใช้ While Loop**
(ทำงานเหมือนเมนู 6 แต่ใช้ While Loop)
- 8: **วนลูปแสดงเลข 1 ถึง N โดยใช้ Do While Loop**
(ทำงานเหมือนเมนู 6 แต่ใช้โครงสร้าง Do While Loop)
- 9: **วนลูปบวกเลข 1 ถึง N โดยใช้ For Loop**
(รับค่าจำนวนเต็ม N จากผู้ใช้ และคำนวณผลรวมของตัวเลขจาก 1 ถึง N)
- 10: **คำนวณผลรวมเลข 1 ถึง N โดยใช้ Recursive Function**
(รับค่าจำนวนเต็ม N จากผู้ใช้ และคำนวณผลรวมของตัวเลขจาก 1 ถึง N โดยใช้ฟังก์ชันเรียกตัวเอง)

3. ออกจากโปรแกรม:

หากผู้ใช้เลือก Q หรือ q โปรแกรมจะแสดงข้อความ "Goodbye!" และปิดโปรแกรม

```

#include <iostream>
#include <string>
using namespace std;

// Regular Function สำหรับการคำนวณพื้นฐาน
double add(double a, double b) {
    return a + b;
}

double subtract(double a, double b) {
    return a - b;
}

double multiply(double a, double b) {
    return a * b;
}

double divide(double a, double b) {
    if (b == 0) {
        cout << "Error: Division by zero!" << endl;
        return 0;
    }
    return a / b;
}

// Recursive Function สำหรับการบวกเลข 1 ถึง N
int sum_recursive(int n) {
    if (n == 0) return 0;
    return n + sum_recursive(n - 1);
}

// Main Function
int main() {
    string username, password;
    cout << "Enter username: ";
    cin >> username;
    cout << "Enter password: ";
    cin >> password;

    if (username != "admin" || password != "admin") {
        cout << "Invalid username or password!" << endl;
        return 0;
    }

    char choice;
    do {
        // แสดงเมนู
        cout << "\nMenu:\n";
        cout << "1. Add Numbers\n2. Subtract Numbers\n3. Multiply Numbers\n";
        cout << "4. Divide Numbers\n5. Calculate Rectangle Area\n";
        cout << "6. Display 1-N (For Loop)\n7. Display 1-N (While Loop)\n";
        cout << "8. Display 1-N (Do While Loop)\n";
        cout << "9. Sum 1-N (For Loop)\n10. Sum 1-N (Recursive)\n";
        cout << "Q/q. Quit\nEnter your choice: ";
        cin >> choice;

        if (choice == '1' || choice == '2' || choice == '3' || choice == '4') {
            double a, b;
            cout << "Enter two numbers: ";
            cin >> a >> b;
            switch (choice) {

```

```

        case '1': cout << "Result: " << add(a, b) << endl; break;
        case '2': cout << "Result: " << subtract(a, b) << endl; break;
        case '3': cout << "Result: " << multiply(a, b) << endl; break;
        case '4': cout << "Result: " << divide(a, b) << endl; break;
    }
    else if (choice == '5') {
        double width, height;
        cout << "Enter width and height: ";
        cin >> width >> height;
        cout << "Area: " << multiply(width, height) << endl;
    }
    else if (choice == '6' || choice == '7' || choice == '8') {
        int n;
        cout << "Enter N: ";
        cin >> n;

        if (choice == '6') {
            for (int i = 1; i <= n; i++) cout << i << " ";
            cout << endl;
        }
        else if (choice == '7') {
            int i = 1;
            while (i <= n) cout << i++ << " ";
            cout << endl;
        }
        else if (choice == '8') {
            int i = 1;
            do {
                cout << i << " ";
                i++;
            } while (i <= n);
            cout << endl;
        }
    }
    else if (choice == '9') {
        int n, sum = 0;
        cout << "Enter N: ";
        cin >> n;
        for (int i = 1; i <= n; i++) sum += i;
        cout << "Sum: " << sum << endl;
    }
    else if (choice == '10') {
        int n;
        cout << "Enter N: ";
        cin >> n;
        cout << "Sum: " << sum_recursive(n) << endl;
    }
    else if (choice == 'Q' || choice == 'q') {
        cout << "Goodbye!" << endl;
    }
    else {
        cout << "Invalid choice. Try again!" << endl;
    }
} while (choice != 'Q' && choice != 'q');

return 0;
}

```

Python

```
def add(a, b):
    return a + b
def subtract(a, b):
    return a - b
def multiply(a, b):
    return a * b
def divide(a, b):
    if b == 0:
        print("Error: Division by zero!")
        return 0
    return a / b

def sum_recursive(n):
    if n == 0:
        return 0
    return n + sum_recursive(n - 1)

def main():
    username = input("Enter username: ")
    password = input("Enter password: ")

    if username != "admin" or password != "admin":
        print("Invalid username or password!")
        return

    while True:
        print("\nMenu:")
        print("1. Add Numbers\n2. Subtract Numbers\n3. Multiply Numbers")
        print("4. Divide Numbers\n5. Calculate Rectangle Area")
        print("6. Display 1-N (For Loop)\n7. Display 1-N (While Loop)")
        print("8. Display 1-N (Do While Loop Emulation)")
        print("9. Sum 1-N (For Loop)\n10. Sum 1-N (Recursive)")
        print("Q/q. Quit")
        choice = input("Enter your choice: ")

        if choice in ['1', '2', '3', '4']:
            a = float(input("Enter first number: "))
            b = float(input("Enter second number: "))
            if choice == '1':
                print("Result:", add(a, b))
            elif choice == '2':
                print("Result:", subtract(a, b))
            elif choice == '3':
```

```

        print("Result:", multiply(a, b))
    elif choice == '4':
        print("Result:", divide(a, b))
elif choice == '5':
    width = float(input("Enter width: "))
    height = float(input("Enter height: "))
    print("Area:", multiply(width, height))
elif choice == '6':
    n = int(input("Enter N: "))
    for i in range(1, n + 1):
        print(i, end=" ")
    print()
elif choice == '7':
    n = int(input("Enter N: "))
    i = 1
    while i <= n:
        print(i, end=" ")
        i += 1
    print()
elif choice == '8':
    n = int(input("Enter N: "))
    i = 1
    while True:
        print(i, end=" ")
        i += 1
        if i > n:
            break
    print()
elif choice == '9':
    n = int(input("Enter N: "))
    total = sum(range(1, n + 1))
    print("Sum:", total)
elif choice == '10':
    n = int(input("Enter N: "))
    print("Sum:", sum_recursive(n))
elif choice.lower() == 'q':
    print("Goodbye!")
    break
else:
    print("Invalid choice. Try again!")
if __name__ == "__main__":
    main()

```

การบ้าน: ระบบล็อกอินพร้อมกำหนดรหัสผ่าน และจำกัดจำนวนครั้งการเข้าสู่ระบบ

รายละเอียดโจทย์:

1. กำหนดชื่อผู้ใช้และรหัสผ่าน:

- เมื่อเปิดโปรแกรม ให้ผู้ใช้กรอก **ชื่อผู้ใช้** และ **รหัสผ่านใหม่** เพื่อบันทึกข้อมูลการเข้าสู่ระบบ (ไม่ต้องเก็บข้อมูลในไฟล์)

2. ล็อกอิน:

- หลังจากตั้งค่าเรียบร้อยแล้ว ให้ผู้ใช้กรอก **ชื่อผู้ใช้** และ **รหัสผ่าน** เพื่อเข้าสู่ระบบ
- หากชื่อผู้ใช้หรือรหัสผ่านไม่ถูกต้อง ให้แจ้งเตือน และลดจำนวนโอกาสในการลองอีก **1 ครั้ง**
- จำกัดจำนวนครั้งในการลองล็อกอินไม่เกิน **3 ครั้ง** หากเกิน 3 ครั้งให้แสดงข้อความ "ออกจากโปรแกรม" และสิ้นสุดการทำงาน

3. เมนูหลัก:

- หลังจากล็อกอินสำเร็จ ให้เข้าสู่เมนูคำนวณและแสดงผลเหมือนโจทย์เดิม:
 - คำนวณพื้นฐาน (บวก, ลบ, คูณ, หาร)
 - คำนวณพื้นที่
 - วนลูปแสดงเลข
 - คำนวณผลรวมเลข

การคำนวณ ข้อ 5 -9 ให้แก้ไขโปรแกรมเป็นการสร้างฟังก์ชัน

4. ออกจากโปรแกรม:

- หากผู้ใช้เลือก Q หรือ q ให้แสดงข้อความ "Goodbye!" และจบโปรแกรม

```

#include <iostream>
#include <string>
using namespace std;

// Function prototypes
double add(double a, double b);
double subtract(double a, double b);
double multiply(double a, double b);
double divide(double a, double b);
int sum_recursive(int n);
void display_numbers_for(int n);
void display_numbers_while(int n);
void display_numbers_do_while(int n);
void sum_numbers_for(int n);

int main() {
    string username, password;

    // Register a new account
    cout << "Register a new account" << endl;
    cout << "Set your username: ";
    cin >> username;
    cout << "Set your password: ";
    cin >> password;

    cout << "\nAccount created successfully! Please log in." << endl;
    string login_username, login_password;
    do {
        cout << "Enter username: ";
        cin >> login_username;
        cout << "Enter password: ";
        cin >> login_password;

        if (login_username == username && login_password == password) {
            cout << "Login successful!" << endl;
            break;
        }
        else {
            cout << "Invalid username or password. Try again." << endl;
        }
    } while (true);

    char choice;
    do {
        cout << "\nMenu:" << endl;
        cout << "1. Add Numbers\n2. Subtract Numbers\n3. Multiply Numbers" <<
endl;
        cout << "4. Divide Numbers\n5. Calculate Rectangle Area" << endl;
        cout << "6. Display 1-N (For Loop)\n7. Display 1-N (While Loop)" << endl;
        cout << "8. Display 1-N (Do While Loop)\n9. Sum 1-N (For Loop)" << endl;
        cout << "10. Sum 1-N (Recursive)\nQ/q. Quit" << endl;
        cout << "Enter your choice: ";
        cin >> choice;

        if (choice == '1' || choice == '2' || choice == '3' || choice == '4') {
            double a, b;
            cout << "Enter two numbers: ";
            cin >> a >> b;

```

```

        switch (choice) {
            case '1': cout << "Result: " << add(a, b) << endl; break;
            case '2': cout << "Result: " << subtract(a, b) << endl; break;
            case '3': cout << "Result: " << multiply(a, b) << endl; break;
            case '4': cout << "Result: " << divide(a, b) << endl; break;
        }
    }
    else if (choice == '5') {
        double width, height;
        cout << "Enter width and height: ";
        cin >> width >> height;
        cout << "Area: " << multiply(width, height) << endl;
    }
    else if (choice == '6') {
        int n;
        cout << "Enter N: ";
        cin >> n;
        display_numbers_for(n);
    }
    else if (choice == '7') {
        int n;
        cout << "Enter N: ";
        cin >> n;
        display_numbers_while(n);
    }
    else if (choice == '8') {
        int n;
        cout << "Enter N: ";
        cin >> n;
        display_numbers_do_while(n);
    }
    else if (choice == '9') {
        int n;
        cout << "Enter N: ";
        cin >> n;
        sum_numbers_for(n);
    }
    else if (choice == '10') {
        int n;
        cout << "Enter N: ";
        cin >> n;
        cout << "Sum: " << sum_recursive(n) << endl;
    }
    else if (choice == 'Q' || choice == 'q') {
        cout << "Goodbye!" << endl;
        break;
    }
    else {
        cout << "Invalid choice. Try again!" << endl;
    }
} while (true);

return 0;
}

double add(double a, double b) {
    return a + b;
}

```



```

double subtract(double a, double b) {
    return a - b;
}

double multiply(double a, double b) {
    return a * b;
}

double divide(double a, double b) {
    if (b == 0) {
        cout << "Error: Division by zero!" << endl;
        return 0;
    }
    return a / b;
}

int sum_recursive(int n) {
    if (n == 0) return 0;
    return n + sum_recursive(n - 1);
}

void display_numbers_for(int n) {
    for (int i = 1; i <= n; i++) {
        cout << i << " ";
    }
    cout << endl;
}

void display_numbers_while(int n) {
    int i = 1;
    while (i <= n) {
        cout << i << " ";
        i++;
    }
    cout << endl;
}

void display_numbers_do_while(int n) {
    int i = 1;
    do {
        cout << i << " ";
        i++;
    } while (i <= n);
    cout << endl;
}

void sum_numbers_for(int n) {
    int sum = 0;
    for (int i = 1; i <= n; i++) {
        sum += i;
    }
    cout << "Sum: " << sum << endl;
}

```

```
def add(a, b):
    return a + b

def subtract(a, b):
    return a - b

def multiply(a, b):
    return a * b

def divide(a, b):
    if b == 0:
        print("Error: Division by zero!")
        return 0
    return a / b

def sum_recursive(n):
    if n == 0:
        return 0
    return n + sum_recursive(n - 1)

def display_numbers_for(n):
    for i in range(1, n + 1):
        print(i, end=" ")
    print()

def display_numbers_while(n):
    i = 1
    while i <= n:
        print(i, end=" ")
        i += 1
    print()

def display_numbers_do_while(n):
    i = 1
    while True:
        print(i, end=" ")
        i += 1
        if i > n:
            break
    print()

def sum_numbers_for(n):
    total = sum(range(1, n + 1))
    print("Sum:", total)
```

```

def main():
    print("Register a new account")
    username = input("Set your username: ")
    password = input("Set your password: ")

    print("\nAccount created successfully! Please log in.")
    while True:
        login_username = input("Enter username: ")
        login_password = input("Enter password: ")

        if login_username == username and login_password == password:
            print("Login successful!")
            break
        else:
            print("Invalid username or password. Try again.")

    while True:
        print("\nMenu:")
        print("1. Add Numbers\n2. Subtract Numbers\n3. Multiply Numbers")
        print("4. Divide Numbers\n5. Calculate Rectangle Area")
        print("6. Display 1-N (For Loop)\n7. Display 1-N (While Loop)")
        print("8. Display 1-N (Do While Loop Emulation)")
        print("9. Sum 1-N (For Loop)\n10. Sum 1-N (Recursive)")
        print("Q/q. Quit")
        choice = input("Enter your choice: ")

        if choice in ['1', '2', '3', '4']:
            a = float(input("Enter first number: "))
            b = float(input("Enter second number: "))
            if choice == '1':
                print("Result:", add(a, b))
            elif choice == '2':
                print("Result:", subtract(a, b))
            elif choice == '3':
                print("Result:", multiply(a, b))
            elif choice == '4':
                print("Result:", divide(a, b))
        elif choice == '5':
            width = float(input("Enter width: "))
            height = float(input("Enter height: "))
            print("Area:", multiply(width, height))
        elif choice == '6':
            n = int(input("Enter N: "))
            display_numbers_for(n)
        elif choice == '7':

```

```
        n = int(input("Enter N: "))
        display_numbers_while(n)
    elif choice == '8':
        n = int(input("Enter N: "))
        display_numbers_do_while(n)
    elif choice == '9':
        n = int(input("Enter N: "))
        sum_numbers_for(n)
    elif choice == '10':
        n = int(input("Enter N: "))
        print("Sum:", sum_recursive(n))
    elif choice.lower() == 'q':
        print("Goodbye!")
        break
    else:
        print("Invalid choice. Try again!")

if __name__ == "__main__":
    main()
```

1.1. ปฏิบัติ (3 ชั่วโมง)

ตัวอย่างที่ 1: โปรแกรม "Hello, World!"

```
#include <stdio.h>

int main() {
    printf("Hello, World!\n");
    return 0;
}
```

ตัวอย่างที่ 2: โปรแกรมคำนวณพื้นที่รูปสี่เหลี่ยม

```
#include <stdio.h>

int main() {
    float length, width, area;
    printf("Enter length: ");
    scanf("%f", &length);
    printf("Enter width: ");
    scanf("%f", &width);
    area = length * width;
    printf("Area of rectangle: %.2f\n", area);
    return 0;
}
```

ตัวอย่างที่ 3: การใช้ฟังก์ชันแยกส่วน

```
#include <stdio.h>

// Function declaration
float calculateRectangleArea(float length, float width);

int main() {
    float length, width, area;

    printf("Enter length: ");
    scanf("%f", &length);

    printf("Enter width: ");
    scanf("%f", &width);

    area = calculateRectangleArea(length, width);

    printf("Area of rectangle: %.2f\n", area);
    return 0;
}

// Function definition
float calculateRectangleArea(float length, float width) {
    return length * width;
}
```